Bachelor's thesis

Degree programme in Information Technology

Internet Technology

2014

Anna Afanasyeva

DEVELOPING A CROWD SIMULATION LIBRARY FOR MOBILE GAMES

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme in Information Technology | Internet Technology

June 2014| 50

Patric Granholm

Anna Afanasyeva

DEVELOPING A CROWD SIMULATION LIBRARY FOR MOBILE GAMES

The purpose of thesis project was to develop a library, later referred to as SteerLib, for realistic crowd simulation in mobile games designed for iOS platform. Today, it is common to see high-end PC and console video games incorporating large heterogeneous crowds, typically consisting of hundreds of NPCs. In contrast, mobile games avoid using crowded environments because of their computational and graphical demands. According to the latest research, modern mobile phones and other mobile devices are capable of simulating large crowds without serious performance degradation. Nevertheless, the preliminary studies made by the author have shown that there is a lack of libraries for simulating large crowds on mobile devices powered with iOS. Thus, SteerLib is intended to help game developers to create iOS games which implement scalable crowds of hundreds of NPCs exhibiting natural behaviour. The library is implemented in Objective-C programming language and is designed to be used with the Sprite Kit framework. The thesis itself is structured as follows: first, it explores the three most common crowd simulation models, which are cellular automata, social forces, and rule-based models. Then, it provides arguments in favor of rule-based models as the most suitable on for implementation in the library. Finally, it discusses the library development and provides test cases.

KEYWORDS:

Crowd Simulation, Artificial Intelligence, Mobile Games, Cellular Automata, Social Forces, Rule-Based Models, Sprite Kit, iOS

CONTENTS

PICTURES

TABLES

LIST OF ABBREVIATIONS (OR) SYMBOLS

2D	2D stands for two-dimensional
3D	3D stands for three-dimensional
AI	AI, which stands for Artificial Intelligence, is an area of computer science that study and design intelligence agents
CPU	CPU, which stands for Central Processing Unit, is the hardware that performs the instructions of a computer program
FPS	FPS, which stands for First Person Shooter, is a video game genre centred on combat through a first-person perspective
GPU	GPU, which stands for Graphics Processing Unit, is the hardware that is designed to accelerate the creation of images on the screen
NPC	NPC, which stands for Non-Playable Character, is a game character that is controlled by the computer through artificial intelligence
RPG	RPG, which stands for Role-Playing Game, is a video game genre in which a player takes on the role of an imaginary character who engage in adventure
TPS	TPS, which stands for Third Person Shooter, is a video game genre in which the player character is visible on-screen and the gameplay consists primarily of shooting

SOURCE CODE

The current version of the SteerLib source code can be obtained from the author by sending an email to anna.afanasyeva@me.com.

1 INTRODUCTION

When taking a look around, it is easy to notice that the real world is a live and crowded place, full of dynamic entities which are constantly and continuously interacting with each other. The most evident examples are cities and villages populated with numbers of people, forests and plains inhabited by flocks of birds and herds of animals, seas and oceans occupied by schools of fish. Even those parts of the world that may, at first glance, look lifeless, such as African deserts or Arctic ice caps are still crowded at the microscopic level. All these facts show that complex and diverse crowds are typical assets of the real world.

Since the rise of video games in the early 1970s, video game developers have had a steadily increasing interest in modelling distinct aspects of the reality in a virtual world. For certain game genres, such as FPS, TPS, RPG, etc., accurate simulating of a lifelike behaviour for groups of autonomous characters, typically known as game AI, appears to be one of the methods to make a game more attractive to a player. However, it is not only the game developers community which has been interested in modelling natural behaviours of groups of people or animals. Most of the crowd behaviour simulation techniques originated from studies on pedestrian flows and evacuation models. Due to vast research in the aforementioned fields, it has been discovered that in a crowd, behaviours of individual entities conform to specific rules, and therefore can be described by mathematical models.

Today, there is a variety of consoles and PC games implementing certain algorithms for simulating crowds. The success of such video game franchises as Assassin Creed, Grand Theft Auto, Dead Rising, StarCraft and Age of Empires is partially determined by amazingly complex behaviours of NPCs that inhabit the virtual worlds of these games. However, modelling a sophisticated appearance and behaviour of large crowds requires significant computational capabilities. As a result, most of the games incorporating crowd simulation techniques are designed to run on high end consoles and PC hardware, while

video games targeted at mobile phones and other mobile devices, so-called mobile games, traditionally avoid using complex and crowded environments.

The worldwide mobile devices market has been rapidly growing and developing in the past few years. High end smartphones and tablets endowed with hardware that is designed for both high performance computation and low power consumption has been released. In particular, in 2013, Apple Inc. announced the fifth generation iPad tablet computer having the following characteristics:

- 64-bit Apple A7 system on chip that includes dual-core CPU with maximum clock rate of 1.3-1.4 GHz and an integrated GPU.
- 1GB of RAM.

At the same time, Apple presented Sprite Kit – a native framework for developing 2D games for iOS and OS X platforms, which allows achieving better performance for games running on Apple's smartphones, tablets, and desktops.

Results from preliminary studies have shown that modern mobile devices having similar hardware configurations are able to support virtual worlds consisting of hundreds of NPCs whose behaviours are governed by certain algorithms (Joselli et al. 2012, 89). As far as the author knows, the Sprite Kit framework lacks libraries implementing realistic crowd behaviour. Therefore, the aims of this thesis are to:

- Select a crowd simulation model that is the most suitable for running on mobile devices.
- Develop a library implementing the selected model in Objective-C programming language to be used with the Sprite Kit framework.

In the thesis, the three following models are considered: cellular automata, social forces, and rule-based models.

2 FUNDAMENTALS

2.1 Definitions and Notations

This subsection briefly explains the terminology used throughout the thesis in order to avoid any possible ambiguity or misunderstanding. First, such essential terms of crowd simulation models as “crowd”, “autonomous agent”, and “emergent behaviour” are discussed. Second, the core aspects of a character’s generation by a means of a computer are briefly stated. Third, the difference between “real-time” and “non-real-time” crowds is clarified.

Crowd (Oxford Dictionaries 2014) *is a large number of people or things gathered together in a disorganized or unruly way.*

In view of the topic of this thesis, the term “crowd” primarily refers to a virtual crowd, i.e., *a crowd generated by a computer*. Members of a virtual crowd are typically called autonomous agents (agent), autonomous character (character), or virtual humans.

An autonomous agent (virtual human, non-playable character) (Davodá 2012, 5) *can be defined as a computer generated and computer-driven representation of a real human, specified by a set of properties, such as geometry, textures, behaviour, position, velocity, direction, etc., which can differ depending on an application.*

In most cases, the terms “agent”, “character”, and “virtual human” are used interchangeably. Particularly in this thesis the term “agent” is primarily used. However, there are three exceptional cases where it is replaced with the more specific ones: when discussing the Reynolds original model (Boid model), the term “boid”, which stands for “bird-like object” is used. In addition, in the Helbing’s social forces model the term “pedestrian” is used, as the model was initially targeted for simulating pedestrian movements. Finally, when speaking of video games, the term “non-playable character” (NPC) is applied.

Generation of a character by a means of a computer consists of the following core aspects (Thalmann and Musse 2013, 3):

- Character animation *is the process of motion which includes collision avoidance in both static and dynamics environments.*
- Character behaviour generation *defines the way the character interacts with other characters and responds to the changes in its surrounding.*
- Character rendering *is the process of displaying animated characters.*

The theoretical part of this thesis focuses on the characters' behaviour generation, omitting all the other aspects. However, in the practical part that includes a game prototype implementing a certain crowd simulation algorithm, all of them are carefully considered.

The collective behaviour of the agents composing a crowd is characterized as an emergent behaviour. Emergent behaviour (Wolfram 2002) *is a behaviour of a complex system that is described by the relationships between the system's individual parts. The main feature of this behaviour is that it cannot be predicted solely based on the system's individual parts, but only by understanding relations between them.*

Virtual crowds are divided into real time and non-real time crowds, depending on the type of the system in which they are implemented.

Real-time system (Oxford Dictionaries 2014) *is a system in which data is processed within milliseconds so that it is available virtually immediately as feedback to the process from which it is coming.*

Video games, as well as most of the pedestrian flow and evacuation simulations, are considered to be real time systems. In a game, a player expects non-playable characters to react accordingly to his or her recent actions. The behaviours of these characters must be calculated in runtime within strict constraints on response time. More requirements and constraints of real time systems are discussed in more details in the following subsection.

In contrast, non-real time systems *do not guarantee immediate feedback, as it is not possible to predict the amount of time that data processing will take.* Movies are examples of non-real time systems. Rendering of realistic virtual characters in a movie scene can take months (Thalmann and Musse 2013, 23) which allows achieving better quality of motion and visuals.

2.2 Requirements and Constraints

When implementing a crowd in real-time applications, such as video games, a number of challenges arises. The two most significant of them are (Thalmann and Musse 2013, 2 - 3):

- the need for efficient variety management and
- the increased demand on computational resources.

In video games, it is common to see non-playable characters that have exactly the same appearance and exhibit very similar behaviour. In contrast, a real life crowd is heterogeneous, i.e., people composing it tend to have unique appearances and unique reactions to the environmental changes. Thus, in order to simulate a naturally looking artificial crowd, it is important to efficiently manage a variety of the character's characteristics at each level of simulation, whether it is rendering, animation, or behaviour generation.

The most constraining factor for simulating crowds in a real-time application is the crowd rendering. The reason is that all the changes in the appearances and behaviour are computed in runtime. As a result, implementing a crowd, consisting of hundreds of unique characters can be computationally and graphically demanding even for modern consoles and computers.

2.3 Microscopic vs. Macroscopic

Different methods exist to classify crowd simulation models according to the properties they have. Subsections 2.3, 2.4, and 2.5 describe the three most commonly used.

Generally, all the crowd simulation approaches can be divided into two main categories: microscopic and macroscopic. Microscopic models focus on the behaviour of individual agents and simulate a crowd as the aggregate of those behaviours. In these models, a set of individual properties, such as position, velocity and so forth, is associated with each agent. The properties directly affect the agent's behaviour at every single moment of time. A wide variety of agents, representing different categories of population, can be described with those parameters. Further, each agent implements a set of simple rules determined by the model. Throughout the simulation time, the agents continuously assess the surrounding environment and each one makes its own decision according to the current situation. Microscopic models allow to simulate complex crowd behaviour, while the behaviour of individual agents remains very simple and homogenous.

The microscopic models can be split into three subcategories (Pelechano et al. 2008, 15): cellular automata models, social force models, and rule-based models. They differ in representation of time and space: cellular automata model implements discrete time and space, while social forces and rule-based models use continuous time and space.

In contrast, macroscopic models do not account for the characteristics and decisions of individual agents, but rather deal with a crowd as a whole. In these models, certain global characteristics are defined in order to describe the global behaviour. Perhaps, the most interesting macroscopic model is the gaskinetics model which represents a crowd as gas or fluid and uses their properties to describe changes in crowd density and velocity (Pelechano et al. 2008, 22).

Microscopic models are typically used to simulate virtual crowds of agents with realistic behaviours; therefore, they will be discussed in greater detail in subsections 3.1, 3.2, and 3.3.

2.4 Discrete-Space vs. Continuous-Space

Another method to categorize crowd simulation models is to group them based on the type of the space they implement. In a discrete-space model, the space is represented by a discrete regular lattice; for instance, a lattice comprising of squares or hexagons. Agents are located at the cells of the lattice, and their coordinates are updated at discrete time steps. A chessboard is a simple real world example of discrete-space models. It consists of 64 square cells which are arranged in eight-by-eight two-dimensional lattice. Each chess piece takes one cell at a time. In a continuous-space model, the space and time are continuous. The real-world itself is typically considered to be a continuous-space model.

2.5 Behavioural vs. Force-Field

The last method to categorize crowd simulation models discussed in this thesis is to group them into behavioural and force-field models (Thalmann and Musse 2013, 11 - 12). In behavioural models, each agent's behaviour is defined using a set of simple rules, which can be considered as an abstract representation of the agent's "psychology". In such a model, the behaviour of an individual agent is influenced by the behaviours of its neighbours. The rule-based models represent the most notable examples of this category. On the contrary, force-field models simulate interactions between agents using analogies with physics; for instance, the social forces model utilizes attraction and repulsion forces to simulate moving agents in a crowd.

2.6 Problem Statement

Today, simulating realistic large crowds is an important goal in scientific, movie, and game communities. Much effort has been put into developing sophisticated approaches to model collective behaviour of agents. The three most commonly used of them are cellular automata, social forces, and rule-based models. Each of these models has certain strengths and weaknesses that need to be carefully

considered before implementation. This thesis mainly focuses on the following important aspects:

- Spatial structure – the structure of a space determines whether the motions of the agents are discretized or not (the concepts of discrete and continuous spaces have been introduced in subsection 2.4). In case of discrete space, the problem of cell size arises. On the one hand, implementing the cell of the size of an agent results in discrete and, therefore, unrealistic movement. On the other hand, when the cell is smaller than the size of the agent, the implementation of the agent's motion becomes more complicated.
- Navigation – coordinating the movements of the agents plays a crucial role in crowd simulations. Each agent in a crowd has an individual target which it wants to reach. If the environment is simple, i.e., not cluttered with obstacles and other agents, the agent can take the straight path to the target. However, in the case of a complex environment, certain path planning techniques are required. A wide variety of such techniques exists for both, discrete and continuous spaces. For continuous space, some high-level representation of the environment is needed. The most popular techniques are potential fields, cell and portal graphs, and roadmaps. For discrete space, A*, potential fields, and flow tiles can be used.
- Behavioural control – in a real world crowd, people are constantly interacting with each other and with the surrounding environment. In order to achieve a more lifelike behaviour for the agents, each crowd simulation model has its own methods and solutions which are either intrinsic or introduced at the development stage.

3 CLASSIC MODELS

The focus of this chapter is to introduce the most relevant crowd simulation models, discuss their strengths and possible weaknesses in the context of their implementation in video games.

3.1 Cellular Automata

This subsection briefly reviews the origin of cellular automata model, explains its fundamentals and application in crowd simulations.

3.1.1 Origin

The concept of cellular automata (sing. cellular automaton) was originally introduced by Neumann and Ulam in the early 1950s. They presented a cellular automaton as a simple mathematical idealization of complex biological systems. In particular, Neumann was interested in modelling self-reproducing organisms (Wolfram 1994, 6). Since that time, the topic has been extensively studied and, as a result, a wide variety of new application areas has been discovered. Today, cellular automata are used in physics to model growth of crystals and flow of fluids; in chemistry – to model dissolutions of particles and enzyme reactions; in biology – to model morphological structures and pigmentation patterns of living organisms (Wolfram 2002). Such a complex phenomenon as crowd behaviour can also be modelled as a cellular automaton.

3.1.2 Fundamentals

3.1.2.1 Definitions

According to Wolfram (Wolfram 1994, 5), cellular automata *are simple mathematical idealizations of natural systems. They consist of a lattice of discrete identical cells, each site taking on a finite set of, say, integer values. The values of the cell evolve in discrete time steps according to deterministic*

rules that specify the value of each cell in terms of the values of neighbouring cells.

Weimar has given a more formal definition of a cellular automaton (Weimar 1997). He specified it in terms of *a tuple* (L, S, N, f)*, where:* L *- is a lattice composed of a set of cells,* S *- is a finite set of states,* S *- is a finite set of neighbouring cells,* $f: S^n \to S$ *– is a state transition function. At each time step, a configuration* $C_t: L \to S$ *associates a state with each cell of the lattice. The following equation is used by the state transition function* f *to change the automaton's current configuration* C_t *into a new configuration* C_{t+1}*:*

$$C_{t+1}(r) = f(\{C_t(i) | i \in N(r)\})$$

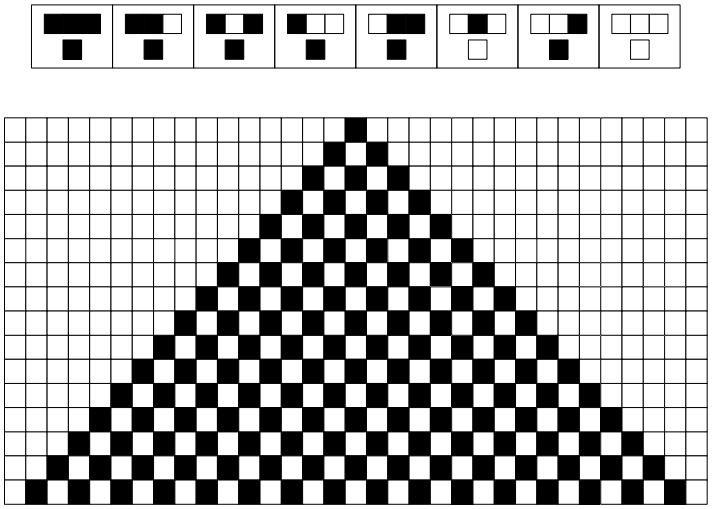
where: r *- is a current cells,* $t \to t + 1$ *- is a discrete time step,* i *- is a single cell,* $N(r)$ *- is the set of neighbours of the cell* r*, defined as:*

$$N(r) = \{i \in L | r - i \in N\}$$

3.1.2.2 Two-Dimensional Cellular Automata

Before considering properties and features of a two-dimensional automaton, a brief overview of the simplest form of cellular automata, a one-dimensional automaton, will be given.

A one-dimensional automaton consists of a line of cells with one or more discrete values in each cell and evolves in discrete steps according to a set of rules, defined by the transition function f, which depend on the state of the cells and the cell's neighbourhood the previous step. Wolfram in his work (Wolfram 2002) has presented a wide variety of rules for a one-dimensional automaton in which each cell has two possible states, black and white. As an example, the representation of the rule 250 is shown on Picture 1.



Picture 1. Rule 250 for one-dimensional automata

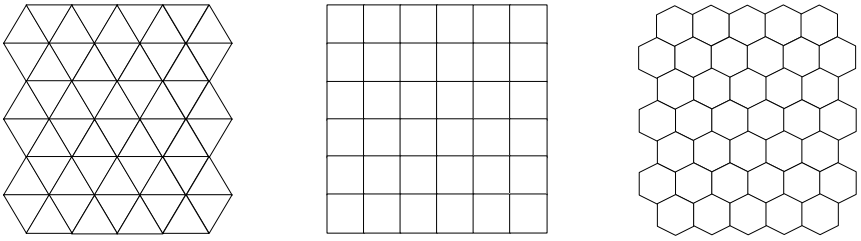
The rule specifies the new colour of the cell (bottom row) for each possible combination of colours of that cell and its immediate neighbours on the previous step (top row).

This example shows that a one-dimensional cellular automaton can produce a complex behaviour based on the underlying basic rules. However, because of its one-dimensional nature, this type of cellular automata is not sufficient for implementing crowd dynamics that usually take place in 2D or 3D Euclidean space. For this purpose, it is common to use two-dimensional cellular automata.

A two-dimensional cellular automaton comprises a lattice of cells and exhibits behaviour similar to the behaviour of the one-dimensional automaton discussed earlier. Generally, it can be characterized by the following features: the lattice geometry, the cell's neighbourhood, and the local transition rule.

3.1.2.2.1 Geometry

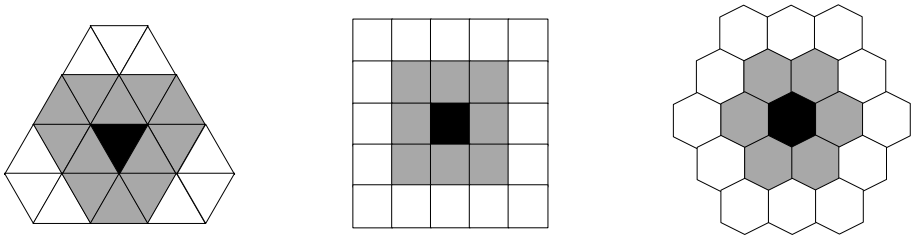
There are three possible types of a lattice a two-dimensional automaton can have: square, triangular, and hexagonal (Picture 2).



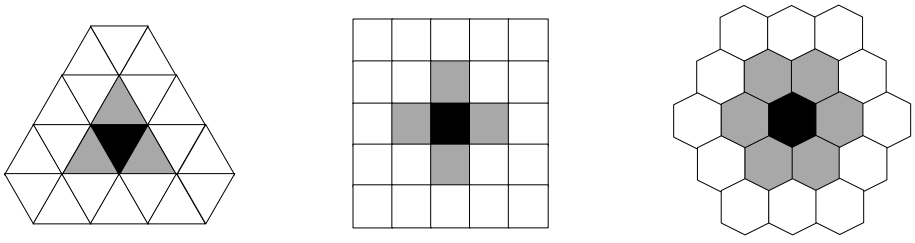
Picture 2. Square, triangular, and hexagonal lattices of a two-dimensional automaton

3.1.2.2.2 Neighbourhood

The two most common neighbourhoods for two-dimensional cellular automata are the Moore neighbourhood and the von Neumann neighbourhood. In the Moor neighbourhood (Picture 3), the central cell is surrounded by a set of orthogonally or diagonally adjacent cell. The von Neumann neighbourhood (Picture 4) comprises only the cells that are orthogonally adjacent to the central cell.



Picture 3. Moore neighbourhood of a cell on different lattices (Nitzsche 2013, 12)



Picture 4. Von Neumann neighbourhood of a cell on different lattices (Nitzsche 2013, 12)

3.1.2.2.3 Transition Rule

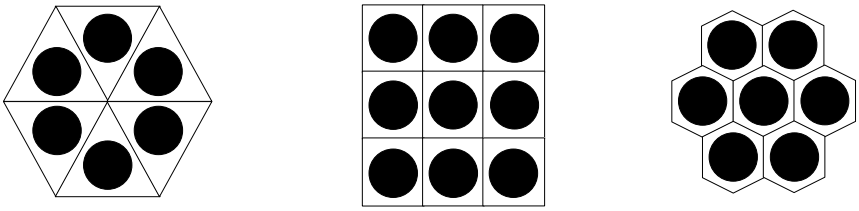
The transition rule, also known as state transition function, is the most important aspect of cellular automata. It ensures a time development from time step $t \rightarrow t + 1$ and determines the evolution of the cellular automaton. Depending on the type of transition rule, all the cellular automata can be divided into synchronous and asynchronous. In a synchronous automaton, the transition rule updates values of all the cells simultaneously. Therefore, a new state of a cell does not influence states of the other cells. In contrast, the transition rule of an asynchronous automaton updates cells one by one so that a new state of a cell affects the states of neighbouring cells. A classical cellular automaton is considered to be a synchronous automaton, as all the cells are updated simultaneously.

3.1.3 Implementation

This subsection considers the cellular automata approach based on the properties discussed in subsection 2.5. Possible limitations of the models and the methods used to overcome them are also presented.

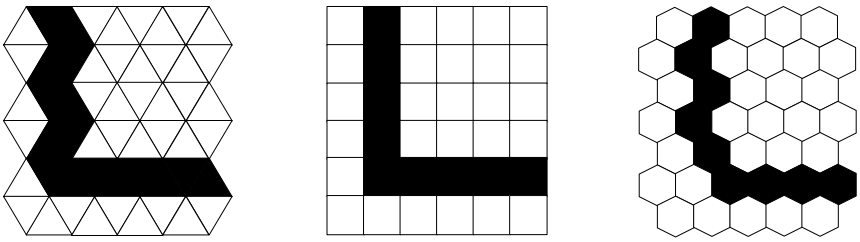
3.1.3.1 Spatial Structure

As mentioned above, two-dimensional cellular automata are the ones that are typically used for modelling crowd dynamics. In this type of cellular automata, the environment is represented by a two-dimensional lattice where each cell, at a time, can either be empty or taken by an agent. Subsection 3.1.2.2.1 briefly described three possible geometries of lattice: triangular, hexagonal, and rectangular. To decide which one to use, it is necessary to consider the placement of agents and obstacles on the lattice (Nitzsche 2013, 11).



Picture 5. High-density agents distribution on different lattices in a two-dimensional automaton (Nitzsche 2013, 11)

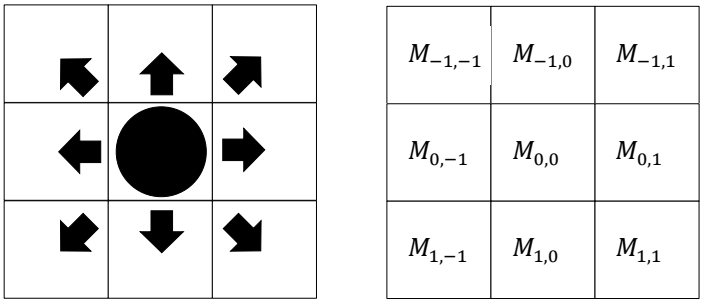
The hexagonal lattice provides a more natural agents distribution. However, it is important to notice that representing hexagonal geometry also requires a larger amount of computational power than representing the rectangular one.



Picture 6. Obstacles placement on different lattices in a two-dimensional automaton (Nitzsche 2013, 11)

The rectangular lattice seems to be more appropriate when implementing walls, while the hexagonal and triangular lattices are more suitable for implementing complex obstacles.

One of the main limitations of cellular automata is the size of a cell. The classical approach in modelling a crowd as a cellular automaton assumes that each agent occupies a single cell at a time step (Picture 7 (a)). Therefore, the size of a cell is considered to be of the size of an agent. Agents move along the lattice by shifting to one of the empty cells in their neighbourhood. This results in a movement comprising discrete displacement of the agent. Consequently, the larger the size of cells, the more unnatural and unrealistic the movements of the agents.



(a) (b)

Picture 7. (a) Possible movement of an agent on a rectangular lattice in a two-dimensional cellular automaton; (b) Matrix of preferences used by an agent in order to find the next move (Burstedde et al. 2001, 511)

The problem of discontinuous movement can be solved using fine grid cellular automata (Sarmady et al. 2010). This method allows to achieve smoother movement by dividing the space into smaller cells and letting each agent to occupy several cells. It decreases the displacement value, and therefore makes the movement look less discrete.

3.1.3.2 Navigation

Two more issues to consider when implementing crowd dynamics are path finding, i.e., finding the shortest path to the destination, and collective behaviour, i.e., interaction between agents, and between agents and the environment. In this subsection a classical cellular automaton is considered, i.e., one agent per cell.

In cellular automata, path finding can be implemented through a lattice-based search using the following methods: A* algorithm, potential fields, and flow tiles. A* search is typically used in video games, while the last two are more common for pedestrian flow and evacuation models. There is a significant difference between these approaches: A* is performed in runtime for each agent in a crowd. In contrast, potential fields or flow tiles use the pre-processed path information, which is stored in the cells.

3.1.3.3 Behavioural Control

Collective behaviour in a cellular automaton can be simulated using the concept of a floor field. The floor filed (Burstedde et al. 2001, 512) is typically represented as a second lattice of cells, either discrete or continuous, overlapping with the main lattice on which the agents are placed. As it was already mentioned, at each time step, an agent can move to one of the neighbouring empty cell. The agent is given a direction of preference, i.e., the direction to the destination point, and a matrix of preferences containing the probabilities for the next move. Picture 7 depicts the possible movements of an agent (a) and an associated matrix of preferences (b). In each step, the agent chooses a desired move according to these probabilities. If the target cell is empty and no other agent targets it, the agent performs the move. If the target cell is occupied, the agent does not move. These probabilities are specified according to the properties an agent has, for instance, velocity. However, as all the agents of the same species share these properties, they also share the matrix of preferences that, in turn, leads to unnatural behaviour. The key idea of using a floor field is that it is modified by the agents which in turn modify the transition probabilities, making the motion more realistic. There are two types of floor fields: static and dynamic.

The static floor field is used to specify the most attractive region of the space, which is the destination point. Such a field depends only on the distance measure to the destination, and therefore does not evolve with time, and is not affected by the presence of the agents. It can be thought as a map that agents use to navigate in space, and avoid the walls and obstacles.

Alternatively, the dynamic floor field is a map of virtual traces left by the agents as their positions changes. It is used to model long-range interaction between them. Such a filed evolves with time. At each time step, each cell diffuses or decays to one of its neighbouring cells that leads to a dilution and finally the vanishing of the traces after some time. This technique is used to avoid moving obstacles and other agents.

In order to simulate collective behaviours more realistically, the number of modification to the concept of the floor field has been introduced (Kirik et al. 2009; Nishinari et al. 2008).

3.1.4 Conclusion on Cellular Automata

The cellular automata model is fast and simple to implement. It offers realistic results for lower density crowds and allows to achieve satisfactory results for high-density crowds using the combinations of the methods described above. This approach can be successfully used in pedestrian flow and evacuation simulations where the quality of motion and visualization is not important. However, unrealistic movement and lack of contacts between agents are serious limitations for the approach to be used for modelling crowds in a video game.

3.2 Social Forces

This subsection considers Helbing's social forces model, discusses its origin, fundamentals, and possible use in crowd simulations.

3.2.1 Origin

In the late 1990s, Helbing and Molnár have proposed a new approach to modelling the motion of pedestrians (Helbing and Molnár 1995; Helbing et al. 2000). According to them, pedestrian flows can be described in terms of "social forces" – a mixture of socio-psychological and physical forces – which are in a sense analogous to Newtonian forces. However, unlike Newtonian forces, social forces are not exerted by the environment, but rather represent the internal motivation of a pedestrian to perform certain actions; for instance, to move in a certain direction. Generally, the social forces model is able to describe the following natural phenomena of pedestrian movement in a crowded environment:

- Each pedestrian normally tries to minimize the effort to reach its destination, so it chooses the fastest route, not the shortest one.

- Each pedestrian prefers to have its individual speed, which depends on such a factors as age, sex, purpose of trip, surrounding, etc.
- Each pedestrian tries to keep a certain distance from the other pedestrians and obstacles.

3.2.2 Fundamentals

The social forces model consists of the following primitives:

- Agent *is a pedestrian that possesses a set of properties, such as coordinates, velocity, interaction with other objects, etc., and is usually represented in the locomotion plane by a circle with its own diameter.*
- Obstacle *is an object on the plane that exerts forces on the agent.*
- Destination *is a location that the agent wants to reach.*

3.2.2.1 Forces

In the social forces model, the motion of pedestrians is fully determined by the following three “social” forces: driving force which reflects the motivation of the pedestrian to move to a certain destination, force of interaction between pedestrians, and force of interaction with obstacles.

3.2.2.1.1 Driving Force

Each individual pedestrian α has a certain destination that he or she wants to reach. The path he or she takes can be described as a polygon with edges $\vec{r}_\alpha^{\,1}, \dots, \vec{r}_\alpha^{\,n} \coloneqq \vec{r}_\alpha^{\,0}$. Therefore, the following equation will define the desired direction $\vec{e}_\alpha^{\,0}(t)$ of movement:

$$\vec{e}_\alpha^{\,0}(t) = \frac{\vec{r}_\alpha^{\,k} - \vec{r}_\alpha(t)}{||\vec{r}_\alpha^{\,k} - \vec{r}_\alpha(t)||}$$

where: $\vec{r}_\alpha(t)$ - is the actual position of the pedestrian at time t, $\vec{r}_\alpha^{\,k}$ - is the desired destination, i.e. the next edge of the polygon to reach.

The driving force itself reflects the motivation of the pedestrian to move in the desired direction $\vec{e}_\alpha^0(t)$ with the desired velocity v_α^0 and is given as:

$$\vec{f}_\alpha^0\,(t) = \frac{1}{r_a}(v_\alpha^0 \cdot \vec{e}_\alpha^0(t) - \vec{v}_\alpha)$$

where: r_α - is a relaxation time of the pedestrian, $\vec{e}_\alpha^0(t)$ - is the desired direction, $\vec{v}_\alpha$ - is the actual velocity.

3.2.2.1.2 Interaction between Pedestrians

In a crowd, each pedestrian normally tries to keep a certain distance to other pedestrians. This behaviour can be described by a repulsive force that depends on the distance between interacting pedestrians as follows: the smaller the distance, the higher the value and vice versa. Therefore, if there are two pedestrians α and β, the repulsive interaction force between them will be:

$$f_{\alpha\beta}(t) = A_\alpha e\left[\frac{r_{\alpha\beta} - d_{\alpha\beta}}{B_\alpha}\right] n_{\alpha\beta}$$

where: A_α and B_α - are constants, $r_{\alpha\beta} = (r_\alpha + r_\beta)$ – is the sum of the pedestrians´ radii, $d_{\alpha\beta} = ||r_\alpha - r_\beta||$ - is the distance between the pedestrians' centres of mass, $n_{\alpha\beta} = \frac{(r_\alpha - r_\beta)}{d_{\alpha\beta}}$ - is the normalized vector pointing from the pedestrian α to the pedestrian β.

In a dense crowd, when physical interaction between pedestrians occurs, i.e., their distance $d_{\alpha\beta}$ is smaller than the sum of their radii $r_{\alpha\beta}$, two additional forces are introduced to the equation: “body force” and “sliding friction force”. “Body force” counteracts body compression reflecting the motivation of a pedestrian to avoid physical damage. It is calculated by the following equation:

$$k(r_{\alpha\beta} - d_{\alpha\beta})n_{\alpha\beta}$$

“Sliding friction force” prevents relative tangential motion reflecting the motivation of a pedestrian to avoid passing close to other pedestrians with a high velocity:

$$k(r_{\alpha\beta} - d_{\alpha\beta})\Delta v_{\alpha\beta}^{t} t_{\alpha\beta}$$

where: $t_{\alpha\beta} = (-n_{\alpha\beta}^{1}, n_{\alpha\beta}^{2})$ – is the tangential direction, $\Delta v_{\alpha\beta}^{t} = (v_{\beta} - v_{\alpha}) \cdot t_{\alpha\beta}$ – is the tangential velocity difference, k and k – are large constants.

The final equation of interaction between pedestrians is:

$$f_{\alpha\beta} = \left\{ A_{\alpha} e \left[\frac{r_{\alpha\beta} - d_{\alpha\beta}}{B_{\alpha}} \right] + kg(r_{\alpha} - d_{\alpha\beta}) \right\} n_{\alpha\beta} + kg(r_{\alpha} - d_{\alpha\beta})\Delta v_{\alpha\beta}^{t} t_{\alpha\beta}$$

where: $g(x)$ – is equal to x if the pedestrians physically interact and is equal to zero if the pedestrians do not touch each other.

3.2.2.1.3 Interaction with Obstacles

The environment typically contains some obstacles, for instance, walls from which a pedestrian tries to keep a certain distance. The force of interaction of a pedestrian α with an obstacle γ can be treated analogous to interaction between two pedestrians:

$$f_{\alpha\gamma} = \left\{ A_{\alpha} e \left[\frac{r_{\alpha} - d_{\alpha\gamma}}{B_{\alpha}} \right] + kg(r_{\alpha} - d_{\alpha\gamma}) \right\} n_{\alpha\gamma} - kg(r_{\alpha} - d_{\alpha\gamma})(v_{\alpha} \cdot t_{\alpha\gamma}) t_{\alpha\gamma}$$

where: $d_{\alpha\gamma}$ – is the distance to an obstacle γ, $n_{\alpha\gamma}$ – is the direction perpendicular to the obstacle, $t_{\alpha\gamma}$ – is the direction tangential to the obstacle.

3.2.3 Implementation

3.2.3.1 Spatial Structure

The social forces model is implemented in a 2D or 3D continuous space that results in a smooth realistic looking motion. The disadvantage of the continuous space approach compared to a discrete space is that updating the position of an agent takes much longer time. Parallel computing is a way to accelerate the performance of this algorithm.

However, path finding algorithms cannot work directly on the space geometry. They require a simplified representation of the environment, usually, in a form of a graph. There is a wide variety of methods that can be used to efficiently represent the environment in which agents move (Pelechano 2006).

The simplest method that works perfectly for flat terrains is to use graphs in which nodes represent available parts of the world and edges represent paths between them. Different methods exist to create such graphs. The most common one is the Voronoi diagram which divides the space into free-spaces where agents can move and spaces taken by obstacles.

Another method of representation of a simple environment has been introduced by Shao (Shao 2005). He proposed to represent environment by a hierarchical collection of the following maps: topological, perception, and path maps. The topological map depicts the topological structure of the virtual world. It is represented as a graph where nodes correspond to the environmental regions and edges represent accessibility between regions. The perception map provides relevant information to perceptual queries. It is a grid map that represents static and dynamic objects, such as obstacles and other agents. Finally, the path map provides the necessary data for implementing path finding, using the Dijkstra or A* search algorithms, or their variations.

For complex environments, including uneven and multi-layered terrains, the approach proposed by Pettré can be applied (Pettré 2005). His idea is based on the standard method where the space is divided into free-space and obstacles. Then the Voronoi diagram of the free space is computed, that is later used to build a set of collision-free convex cells. The navigation graph is obtained for the adjacency graph of the cell. The novelty of this work is to classify the part of the terrain into free spaces and obstacles according to the slope angle.

3.2.3.2 Navigation

When the space has been partitioned and represented in a form of a graph using the methods described above, different types of path planning algorithms

can be then applied in order to find the shortest path to the destination. Examples of such algorithms are the Dijkstra and an improvement over Dijkstra which is called A*.

However, for navigation we can also use methods that do not require pre-partitioning, such as probabilistic roadmaps (Kavraki et al.1996) method which is commonly used in robotics. This method consists of two phases: a construction phase in which a roadmap represented by an undirected graph is built, and a query phase in which the Dijkstra algorithm is used to obtain the shortest path between the initial position and the destination.

3.2.3.3 Behavioural Control

The generalized Helbing's model can be extended to include some more individualism. Braun and Musse in their work (Braun and Musse 2003) noticed that in different situations people can react in different ways depending on their individual characteristics and on group structure. The model they proposed included such characteristics of an individual agent as a level of altruism, a need for help, and an identifier if the agent belongs to a family which is a predefined group of agent formed by some agent who know each other. Generally, Braun and Musse's work allows to create a more realistic simulation of crowd behaviour.

3.2.4 Conclusion on the Social Forces Model

The social forces model can simulate a wide variety of natural effects found in crowd behaviour, such as collision avoidance, jamming at exits, and virtual roundabouts in places where two flows intersect. It offers a simple way to handle interaction between agents and obstacles using only attractive and repulsive forces. Generally, it fits well for implementing a crowd; however, in an environment cluttered with obstacles and other agents, the model can become quite CPU-intensive, as it will require to calculate all the interactions with obstacles and agents. Moreover, in a cluttered environment, where many different forces act simultaneously on the agent, the problem of agent shaking

can appear and that significantly decreases the level of realism of the simulation.

3.3 Rule-Based Models

There is a variety of behavioural rules (Pelechano et al. 2008, 20) that can be applied to individuals or small groups in order to achieve more realistic crowd behaviour. Perhaps, the most well-known of them are the Reynolds rules for simulating flocking behaviour, also referred to as the Boids model (Reynolds 1987). The origin of the model, its basic theory, expansions, and possible use in crowd simulations are discussed in this subsection.

3.3.1 Origin

In 1987, Craig Reynolds presented a new approach to modelling a complex group behaviour of self-organizing entities. His research originated from observations of birds in flight and later led to the development of the flocking model, which is also known as the Boid model. This model is the subject of the following subsection.

3.3.2 Fundamentals

3.3.2.1 Definitions

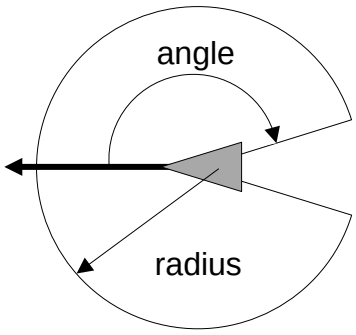
Each boid in the Reynolds Boid model is represented as a 2D or 3D space point particle defined by a position, velocity, perception radius, and mass properties. Such a particle is also referred to a point of mass (Reynolds 1987). *The mass of the boid is an assigned property, while the position and velocity are calculated by using a combination of steering behaviours which will be introduced later in the subsection.*

When implementing flocking behaviours, certain information about a boid's neighbourhood is required. In his early paper (Reynolds 1987), Reynolds defined the boid's neighbourhood as all the other boids within a simple perception sphere. However, in the later (Reynolds 1999) a concept of a "field

of view" was introduced as an attempt to represent a natural bird's field of vision. The "field of view" is merely an arc, which is specified by the two following quantities:

- Angle – defines boid's "field of view". Generally, a wide field of view allows to simulate a relatively large flock of birds, while a narrow field of view results in a group of boids organized in a single line;
- Radius – defines whether two boids are nearby. The higher the value of the radius, the more entities are visible to the boid which results in a more cohesive group.

When applying flocking behaviours, only the boids within a neighbourhood are considered in calculations; all the others are simply ignored.



Picture 8. Neighbourhood of a boid

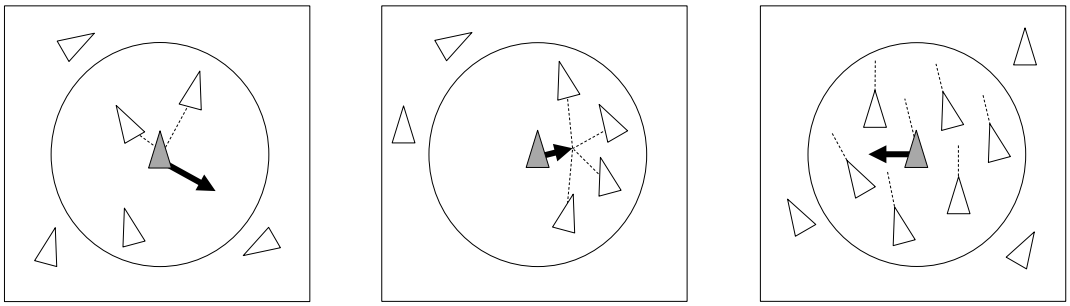
3.3.2.2 Original Model

The Reynolds flocking model assumes that the flock is the result of the interaction between the behaviours of individual bird (boids). Thus, its aggregate motion can be created by a distributed behavioural model where each bird is implemented as an independent entity which navigates in the dynamic environment using the combination of perception mechanisms with a set of simple rules governing its behaviour. In the basic model, these rules are stated as follows (Reynolds 1987, 6):

- Separation – steer to avoid collision with local flockmates. Each boid tries to maintain a certain distance to others nearby.

- Cohesion – steer to move towards the average position of local flockmates. Each boid tries to stay close to other boids.
- Alignment – steer towards the average heading of local flockmates. Each boid tries to match the nearby boids in direction and velocity.

Initially, he referred to separation, cohesion, and alignment as to collision avoidance, flock centring, and velocity matching accordingly.



(a) (b) (c)

Picture 9. Steering behaviour rules for boid in Reynolds Boid model: (a) Separation, (b) Cohesion, (c) Alignment

Later in 1999, Reynolds has introduced a more general concept of steering behaviours that enhanced the behaviours already present in the original model by adding new rules (Reynolds 1999). They are the subjects of the following subsection.

3.3.2.3 Enhanced Model

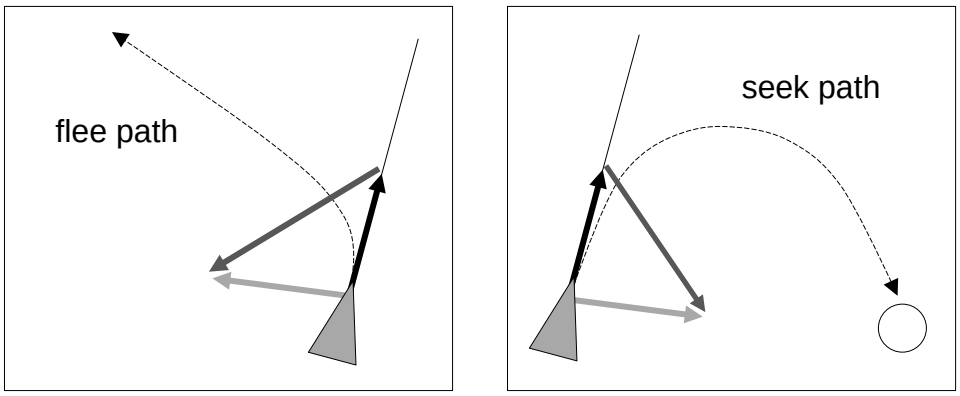
In this thesis, the following five behaviours will be considered: seek and flee, wander, arrive, collisions and obstacle avoidance, and leader following.

3.3.2.3.1 Seek and Flee

Seek behaviour attracts an agent to a specific static position in the environment at the greatest possible acceleration. Picture 10 (b) illustrates the path that results from this behaviour and the forces acting on the agent. The black vector shows the agent´s current velocity. The grey vector represents the “desired velocity” and points in the direction from the boid to the target. Depending on a

particular application, the length of this vector can be either equal to the agent´s maximal speed or to the agent's current speed. The dark grey vector is the difference between the agent´s current velocity and the agent´s "desired velocity". The dotted line shows the path taken by the agent influenced by the forces.

Flee is an inverse of the seek behaviour. It acts to get the agent as far away from the target as possible. On Picture 10 (a), the dotted line, the black and grey vectors depict the path, current velocity, and "desired velocity" accordingly. The dark grey vector similarly to the dark grey vector on Picture 10 (b) shows the difference between the current and "desired" velocities.

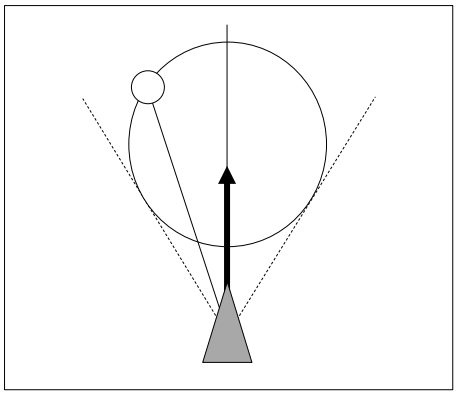


Picture 10. (a) Seek behaviour of an agent, (b) Flee behaviour of an agent

Seek and flee behaviours are defined only for static targets; however, they are not sufficient when the agent is chasing a moving target or is escaping from a chaser. To implement interactions with a dynamic target, pursue behaviours and its opposite, evade behaviour, are used. The main problem with pursue is that the effective behaviour requires the prediction of the next position of the moving target. One of the simplest approaches to predict the next movement is to simulate the next step based on the agent´s current velocity and position. Then seek can be simply reapplied for the predicted position in order to get pursuit.

3.3.2.3.2 Wander

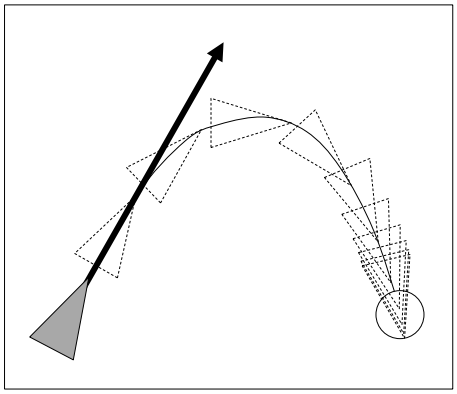
Wander is a type of steering behaviour that controls the agent moving aimlessly around. It can be achieved either by generating random steering forces at each time step or by keeping the agent's general steering direction with small displacements at each time step. The second approach produces smooth movement, while the first approach produces some linear and angular jerkiness that makes movement less natural. Picture 11 shows one of the possible implementations of the second approach. The wander direction of the agent is represented by the small circle, which is constrained to lie on the edge of the large circle. At each time step, a random offset is added to the wander direction, that forces the agent to smoothly change its direction. The black vector on the picture represents velocity.



Picture 11. Wander behaviour of an agent

3.3.2.3.3 Arrival

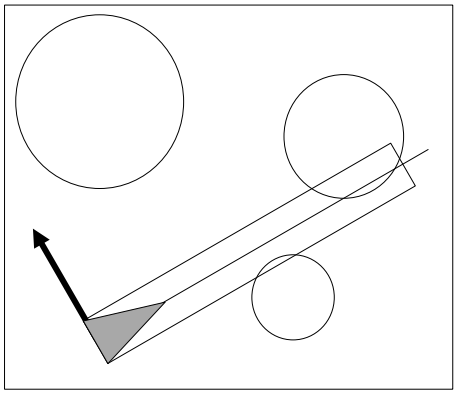
The arrival behaviour similarly to the seek behaviour attracts the agent to its target location. However, as it was discussed above, seek steers the agent through the target at the full speed. In contrast, arrival behaviour causes the agent to slow down so that it arrives exactly at the right location. Picture 12 shows the arrival behaviour. the black arrow represents velocity, the white circle is the target location.



Picture 12. Arrival behaviour of an agent

3.3.2.3.4 Collisions and Obstacles Avoidance

When agents move in a cluttered environment, full of other agents and obstacles, a special type of behaviour for avoiding constant collisions is needed. The collision avoidance behaviour gives to the boid the ability to manoeuvre in such an environment. Compared to the flee behaviour that always causes the agent to steer from a given location, the collision avoidance behaviour is only engaged if the obstacle lies in front of the agent. For the sake of simplicity, both the agent and the obstacle are approximated as a sphere although more complex shapes can be also used.

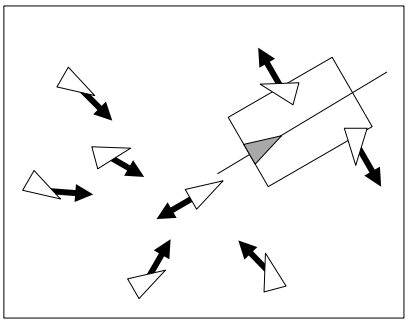


Picture 13. Obstacles avoidance behaviour

3.3.2.3.5 Leader Following

Besides from moving as a group, agents can have a leader to follow. Picture 11 represents this kind of behaviour where the white agents follow the grey agent. Generally, the behaviour of the white agents, also called followers, is controlled by the following rules:

- Stay near the leader without crowding the leader – the implementation of this rule relies on the arrival behaviour. The target point, which all the followers desire to move is placed slightly behind the leader.
- Stay out of the leader's way – the followers avoid the rectangular area in front of the leader and to steer away if they found themselves in this area, before continuing the arrival behaviour.
- Avoid colliding with each other – this rule uses separation behaviour to prevent agents bumping into each other.



Picture 14. Leader following behaviour

3.3.2.3.6 Combining Steering Behaviours

In a video game, an NPC usually needs more complex behaviour than simply moving toward the goal or evading from a danger. It needs to avoid collisions with other characters and walls, while moving towards its goal, tends toward safety as it moves, and so on. Such a complex behaviour can be achieved by combining together basic steering behaviours. There are two methods for combining behaviours: arbitration and blending (Reynolds 1999; Millington and Funge 2009, 96).

In the arbitration method, the boids switch between different steering behaviours according to the changes in the environment. For instance, in a shooter game, an NPC can chase the player, but when it is out of ammo, it will run away.

In the blending method, several steering behaviours are blended together in order to act in parallel. For instance, an NPC can chase the player at the same time avoiding collision with obstacles. There are several approaches to blending steering behaviours. The most straightforward of them is to compute the component steering behaviours and sum them together with applying a weighting factor for each of them. This approach is known as “weighted blending”. It is easy and fast to implement; however, it has two significant weaknesses that have to be taken into consideration: weighted blending is not computationally efficient, and in some cases components may cancel each other.

3.3.3 Implementation

3.3.3.1 Spatial Structure

Similarly to the social forces model, the Reynolds rule-based model uses a continuous 2D or 3D Euclidean space and relies on the environment representation methods described in subsection 3.2.3.1.

3.3.3.2 Navigation

The Reynolds model is a continuous-space model, therefore the methods described in subsection 3.2.3.2 can be used for navigation.

3.3.3.3 Behavioural Control

The Reynolds rule-based model allows to apply different behavioural rules to individuals or groups in order to achieve more believable overall behaviour.

3.3.4 Conclusion on the Rule-Based Model

The Reynolds model allows to simulate complex crowd behaviour combining basic steering behaviours. The advantage of the model is that there is no contact between agents, therefore, no need to calculate collision detection and response. However, special algorithms need to be used to achieve the desired behaviour; also, fine tuning the behaviour can be a time-consuming activity.

3.4 Model Comparison

This subsection compares the crowd simulation models discussed in the thesis. Table 1 emphasizes the main features of the models in order to select the one most appropriate to be implemented in the library.

Table 1. Comparison of different crowd simulation models

	Spatial structure	Collision response	Agent shaking	Agent overlapping	Parallel implementation	Level of realism	Ease of implementation
Cellular Automata	Discrete	Yes	No	No	Possible	Low	Easy
Social Forces (Helbing)	Continuous	No	Yes	No	Possible	Medium	Medium
Rule-Based Models	Continuous	No	No	No	Possible	High	Easy

3.5 Conclusion on Classic Models

The Reynolds steering behaviours model seem to be the most appropriate model for implementation of a mobile device. Despite the fact that it has some disadvantages which are described in details in subsection 3.3, it allows to produce realistic behaviour of the individuals in a crowd and the crowd as a whole. Further, steering behaviour can be implemented on multiprocessor systems the same easy as on single processor systems. The following chapter describes the implementation of the Reynolds steering behaviours model in a library to be used on mobile devices powered with iOS.

4 IMPLEMENTATION

4.1 Sprite Kit and SteerLib

In 2013, Apple Inc. released Sprite Kit which is a framework for developing 2D games for iOS and OS X platforms. Sprite Kit provides the following functionality: graphics rendering and animation infrastructure, means for creating complex special effects, integrated physics simulation, and sound playback support. Moreover, the fact that it is a proprietary framework indicates that video games developed with Sprite Kit are optimized to work on Apple hardware.

SteerLib is a library designed to be used with the Sprite Kit framework in order to simplify the process of development of realistic crowds. The library implements the Reynolds steering behaviours model for simulating lifelike behaviour of large groups of NPCs. In the current version of SteerLib, the following behaviours are available to an NPC:

- Flocking, which includes separation, cohesion, and alignment.
- Seek and flee.

4.2 Architecture

In this subsection, the architecture of the library is discussed. First, implementation of a boid is considered. Then, the algorithms used in steering behaviours are explained. In order to avoid unnecessary complexity, all the algorithms are presented in pseudocode.

4.2.1 Agent

Each agent is specified by a set of properties, which are directly coupled with its behaviours, thus affecting the final motion. The values they take can vary from agent to agent which allows to create more realistic behaviours of the crowd. Particularly, in SteerLib an agent possesses the following properties:

- location – is a vector in 2D space that defines the agent's coordinates;
- velocity – is a vector in 2D space that determines the agent's speed and direction;
- acceleration – is a vector in 2D space that defines the agent's acceleration calculated based on the components of the steering force applied to it;
- maxSpeed – defines the maximum speed of the agent;
- maxAcceleration – defines the maximum acceleration of the agent;
- neighbourRad – defines the neighbourhood radius;
- separationDist – defines the separation distance between agent and is used to calculate separation force;
- wanderOrient – defines the orientation of the wander target;
- wanderRate – defines maximum rate at which the wander orientation can change.

4.2.2 Neighbourhood

A variety of algorithms of different complexity exists to implement the neighbourhood of an agent in a flock. The library applies the classical Reynolds neighbourhood, which is specified by all the agents within a certain radius. The possible implementation is shown in Algorithm 1. The algorithm compares the distances of all the pairs of agents in the crowd in order to find neighbouring ones. If such agents are found, they are added to the agent's neighbourhood.

Algorithm 1. Neighbourhood gathering

```
method getNeighbours
foreach iAgent in crowd do
    foreach jAgent in crowd do
        if  iAgent != jAgent AND
        distBetween (iAgent, jAgent) < neighbourRad then
            neighbourhood = neighbourhood + jAgent
        end if
```

end foreach

end foreach

This algorithm, which is also known as a brute force algorithm (Joselli et al. 2012) has a complexity of $O(n^2)$, where n is a number of agents in a crowd. It has been chosen for its simplicity; however, computation of the algorithm is not efficient and can be improved.

4.2.3 Behaviours

At the present time, SteerLib implements three basic flocking behaviours (separation, cohesion, and alignment) and two additional ones (seek and flee). This subsection is structured as follows: first, Algorithm 2, Algorithm 3, and Algorithm 4 implement the basic flocking behaviours. Then, Algorithm 5 and Algorithm 6 implement the additional steering behaviours. Finally, Algorithm 7 and Algorithm 8 calculate the final acceleration using weighted blending method described in subsection 3.3.2.3.6.

To calculate the steering acceleration produced by the separation behaviour (Algorithm 2), for each agent in the crowd, the relative distance to its neighbours is computed. If the distance is smaller than the separation distance, meaning that the agents are too close to each other, a normalized weighted by distance vector pointing away from the neighbour is calculated.

Algorithm 2. Calculating Separation behaviour acceleration

method separation(iAgent)

vector = (0, 0)

neighbourhood = getNeighbours(iAgent)

foreach jAgent in neighbourhood **do**

if distance(iAgent, jAgent) < separationDist **then**

vector = vector +((locationOf(iAgent) – locationOf(jAgent)

normalize(vector)

```
        vector = vector / distance(iAgent, jAgent)
    end if
end foreach
if (sizeOfNeighbourhood > 0) then
    vector = vector / sizeOfNeighbourhood
return vector
```

To compute the cohesion behaviour acceleration (Algorithm 3), for each agent in the crowd, the group centre position is calculated by averaging the positions of its neighbours. Then, the agent steers to that position using the method steerTo().

Algorithm 3. Calculating Cohesion behaviour acceleration

```
method cohesion(iAgent)
target = (0, 0);
neighbourhood = getNeighbours(iAgent)
foreach jAgent in neighbourhood do
        target = target + locationOf(jAgent)
end foreach
if (sizeOfNeighbourhood > 0) then
    target = target / sizeOfNeighbourhood
return steerTo(target)
```

Alignment behaviour acceleration is calculated according to the Algorithm 4. The algorithm simply matches the agent's velocity with to the velocities of its neighbours.

Algorithm 4. Calculating Alignment behaviour acceleration

```
method alignment(iAgent)
vector = (0, 0)
```

```
neighbours = getNeighbours(iAgent)
foreach jAgent in neighbours do
        vector = vector + (velocityOf(iAgent) – velocityOf(jAgent))
end foreach
if (sizeOfNeighbourhood > 0) then
    vector = vector / sizeOfNeighbourhood
return vector
```

The described algorithms have complexity $O(n)$, where n is the number of agents in the neighbourhood. This approach is efficient for a small number of characters on a game level. For a few hundreds characters, a faster method is required (Millington and Funge 2009).

Seek acceleration (Algorithm 5) is calculated by merely matching the position of the agent with the position of its target. The performance of the algorithm is $O(1)$ (Millington and Funge 2009).

Algorithm 5. Calculating Seek behaviour acceleration

```
method seek(iAgent, jAgent)
vector = (0, 0)
vector = locationOf(jAgent) – locationOf(iAgent)
normalize(vector)
vector = vector * maxAcceleration
return vector
```

As flee is the opposite of seek, its acceleration is calculated by merely flipping the order of terms in the third line of the code (Algorithm 6). The performance of the algorithm is again $O(1)$ (Millington and Funge 2009).

Algorithm 6. Calculating Flee behaviour acceleration

```
method flee(iAgent, jAgent)
vector = (0, 0)
vector = locationOf(iAgent) – locationOf(jAgent)
normalize(vector)
vector = vector * maxAcceleration
return vector
```

To calculate the final acceleration of the agent, SteerLib uses the weighted blending method. The final acceleration for each agent is calculated by summing up the components of the steering behaviours which are multiplied by weighting factors. Algorithm 6 shows the calculation of the final acceleration for separation, cohesion, and alignment flocking behaviours.

Algorithm 7. Calculating final acceleration with the weighted blending method

```
    foreach agent in crowd do
        separation = separation(agent) * separationWeight
        cohesion = cohesion(agent) * cohesionWeight
        alignment = alignment(agent) * alignmentWeight
        return separation + cohesion + alignment
    end foreach
```

At each time step, final acceleration is computed for each agent in the crowd. Then, the acceleration is added to velocity, which is limited by maximum velocity to avoid the agent to move too fast. The new velocity is then added to the agent location to translate it on the map (Algorithm 6). Algorithm 6 shows the main loop of the game where the position of the agent is updated.

Algorithm 8. Main game loop

```
foreach timeStep do
```

```
    acceleration = agentAcceleration
    velocity = velocity + acceleration
    location = location + velocity
    draw(agent)
end foreach
```

5 CONCLUSION

The aim of this thesis was to develop a library for fast and easy implementation of realistic crowd behaviour in iOS mobile games that can be used with the Sprite Kit framework. The three most commonly applied crowd simulation models, which are cellular automata, social forces, and rule-based models, have been discussed. As a result, the Reynolds rule-based model has been selected as the most effective one. The selection has been made based on the criteria summarized in Table 1 in subsection 3.4.

SteerLib library has been implemented using the Objective-C programming language. The library computes the three basic steering behaviours: separation, cohesion, and alignment. Further, two additional behaviours have been implemented, which are seek and flee.

The performance of the code is an important criterion of the evaluation. It is mainly depends on the crowd size and the number of agents in a neighbourhood. The testing performed on an emulator of iPhone 5s powered with iOS 7 demonstrated that the crowd consisting of 500 agents is rendered at a rate of 50 FPS.

REFERENCES

Braun, A.; Musse, S. R.; deOliveira, L. P. L. & Bodmann, B. E. J. 2003. Modeling individual behaviors in crowd simulation. Proceedings of Computer Animation and Social Agents (CASA), IEEE Computer Society, 143–148.Burstedde, C.; Klauck, K.; Schadschneider, A. & Zittartz, J. 2001. Simulation of pedestrian dynamics using a two-dimensional cellular automaton. Physica A: Statistical Mechanics and its Applications Vol. 259, 507-525. Consulted 22.05.2014 http://arxiv.org/pdf/cond-mat/0102397.pdf?origin=publicationDetail.

Davodá J. 2012. Cellular automata approach for crowd simulation. Rigorous thesis. Bratislava: Comenius University.

Helbing, D.; Farkas, I. & Vicsek, T. 2000. Simulating dynamical features of escape panic. Munich: Nature Publishing Group.

Helbing, D. & Molnár, P. 1995. Social force model for pedestrian dynamics. Physical review E Vol. 51, 4282. Consulted 22.05.2014 http://arxiv.org/pdf/cond-mat/9805244.pdf.

Joselli, M.; Passos, E. B.; Silva, J. R. J.; Zamith, M; Clua E. & Soluri, E. 2012. A flocking boids simulation and optimization structure for mobile multicore architecture. Proceedings of Brazilian symposium on computer games and digital entertainment (SBGames 2012). Consulted 22.05.2014 http://base.gamux.com.br/events/2012.11.02-SBGames12/ proceedings/papers/ computacao/comp-full_11.pdf.

Kavraki, L. E.; Svetska, P.; Latombe, J.-C. & Overmars, M. H. 1996. Probabilistic for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation Vol. 12 No. 4/1996, 566-580.

Kirik, E.; Yurgel'yan, T. & Krouglov, D. 2007. An intelligence floor field cellular automaton model for pedestrian dynamics. Proceedings of the 2007 Summer Computer Simulation Conference. Consulted 22.05.2014 http://crowd.krasn.ru/st/nstat3.pdf.

Millington, I. & Funge, J. 2009. Artificial intelligence for games. Burlington: Morgan Kaufmann Publishers.

Nishinari, K.; Kirchner, A.; Namazi, A.; Schadschneider, A. 2004. Extended floor field CA models for evacuation dynamics. IEICE Transactions on Information and Systems Vol. E87-D No. 3/2004, 726-732.

Nitzsche, C. 2013. Cellular automata modeling for pedestrian dynamics. Bachelor thesis. Greifswald: University of Greifswald.

Oxford Dictionaries 2014. Consulted 22.5.2014 http://www.oxforddictionaries.com.

Pelechano, N. 2006. Modeling realistic high density autonomous agent crowd movement: social forces, communication, roles and psychological influence. Dissertation. Pennsylvania: University of Pennsylvania.

Pelechano, N; Allbeck, J. & Badler N. 2008. Virtual crowds. Methods, simulation, and control. California: Morgan & Claypool Publishers.

Pettré, J.; Laumond, J.-P. & Thalmann, D. 2005. A navigation graph for real-time crowd animation on multilayered and uneven terrain. First International Workshop on Crowd Simulation Vol. 43, 194. Consulted 22.05.2014 http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.1573&rep=rep1&type=pdf.

Reynolds, C. 1987. Flocks, herds, and schools: a distributed behavioral model. Proceedings of the 1987 ACM SIGGPAPH Vol. 21, 25-34.

Reynolds, C. 1999. Steering behaviors for autonomous characters. Proceedings of Game Developer Conference Vol.1999, 763–782.

Sarmady, S.; Haron, F. & Talib A.Z. 2010. Simulating crowd movements using fine cellular automata. Proceeding of 12th international conference on computer modeling and simulation (UKSim), 428-433.

Schiff, L. J. 2008. Cellular automata: a discrete view of the world. New Jersey: John Wiley & Sons.

Shao, W.; Terzopoulos, D. 2005. Autonomous pedestrians. Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation, 19-28.

Thalmann, D. & Musse S. 2013. Crowd simulations. Second edition. London: Springer.

Weimar, J. R. 1997. Simulation with cellular automata. Berlin: Logos-Verlag.

Wolfram, S. 1994. Cellular automata and complexity. Collected papers. Boulder: Westview Press.

Wolfram, S. 2002. A new kind of science. Champaign: Wolfram Media.